Foundations of Computation

Lecture 6: Regular expressions

Thomas Powell

19 October 2023

Regex and search patterns

You have probably used regular expressions before. They allow us to search for and identify *patterns* in strings e.g. email addresses:

$$[a-zA-z0-9]^*@[a-zA-z0-9]^*.(com \cup co.uk)$$

Regular expressions have a vast range of applications, including:

- Search and replace in a text editor.
- Validating user input in an application.
- Extracting information from a webpage e.g. topics, links, email addresses (can be done on a very large scale).
- Data processing and analysis (might want to find instances of a certain thing, eliminate noise).
- Searching for file types on an operating system.
- Syntax highlighting in an IDE.
- Lexical analysis as the first step of a compiler: Transforming a sequence of characters into a sequence of tokens.

Regular expressions describe patters

- **Pattern:** *"Some zeros, then a one, then either a zero or a one, then finally a zero"*

  Regex: $0^*1(0 \cup 1)0$

- **Pattern:** *"either the empty string, or an even number of ones followed by a single zero"*

  Regex: $e \cup (11)^*0$

- **Regex:** $(010)^* \cup (00 \cup 11)^*$

  Pattern: ?

Regular expressions

Definition

The set $REG(\Sigma)$ of regular expression over an alphabet Σ is defined by the following inductive rules:

1. $a \in REG(\Sigma)$ for any $a \in \Sigma$
2. $e \in REG(\Sigma)$
3. $\emptyset \in REG(\Sigma)$
4. $(\alpha \cup \beta) \in REG(\Sigma)$ for any $\alpha, \beta \in REG(\Sigma)$
5. $(\alpha\beta) \in REG(\Sigma)$ for any $\alpha, \beta \in REG(\Sigma)$
6. $(\alpha^*) \in REG(\Sigma)$ for any $\alpha \in REG(\Sigma)$

Note. We typically remove brackets, using the convention that $*$ binds closer than concatenation, which in turn binds closer than union. So

$$ab^* \cup ba^* = ((a(b^*)) \cup (b(a^*)))$$

The language associated with a regular expressions

Definition

Any regular expression $\alpha \in REG(\Sigma)$ represents a language $L(\alpha) \subseteq \Sigma^*$ using the regular operations as follows:

1. $L(a) = \{a\}$
2. $L(e) = \{e\}$
3. $L(\emptyset) = \emptyset$ (empty language)
4. $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$ (union)
5. $L(\alpha\beta) = L(\alpha)L(\beta)$ (concatenation)
6. $L(\alpha^*) = L(\alpha)^*$ (Kleene star)

Examples

- $L(ab^* \cup b^*a)$

  Words over $\{a, b\}$ of the form ab^n or b^na for $n \geq 0$.

- $L((ab(a \cup b))^*)$

  Words over $\{a, b\}$ made up of sequences of blocks in $\{aba, abb\}$ e.g. $e, aba, abbaba$.

- Words over $\{a, b\}$ containing an even number of as.

  $L((b \cup ab^*a)^*)$

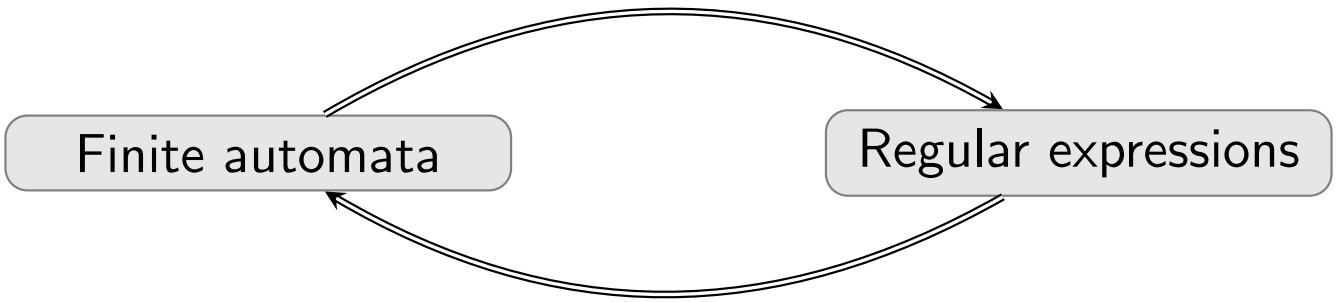
Putting everything together

We have defined regular expressions, and show that these give rise to a class of languages $L(\alpha)$ represented by regular expressions. Coincidentally, if a language is accepted by a finite automaton it is called "regular"...

Theorem

The regular languages are exactly those languages that can be represented by a regular expression.

This is the biggest result of the unit so far! It connected the abstract world of **finite automata** with the practical world of **regular expressions**.

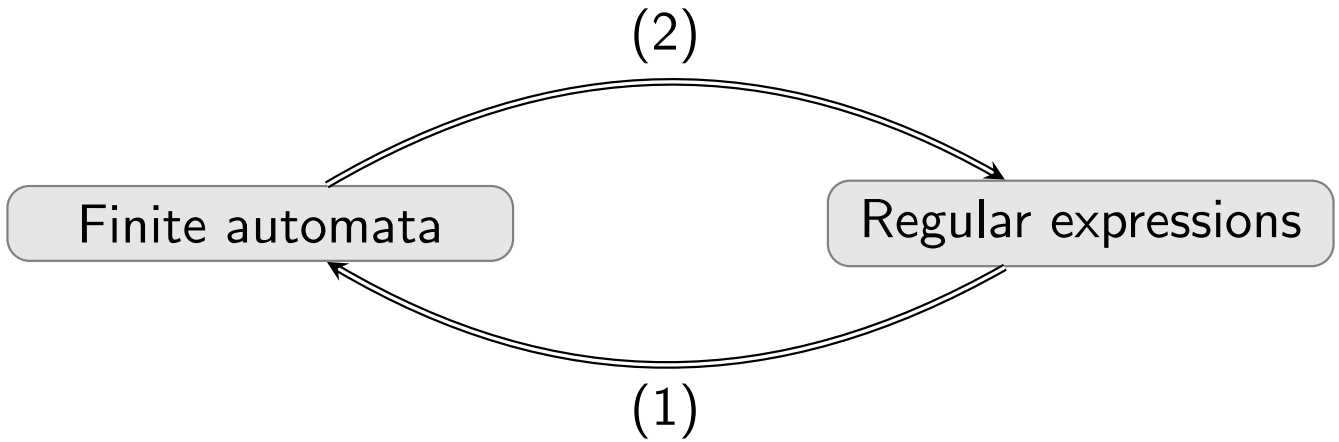


How do we prove the theorem?

It suffices to show the following two things:

1. For any regular expression α the language $L(\alpha)$ is regular i.e. accepted by some DFA/NFA (easy direction).

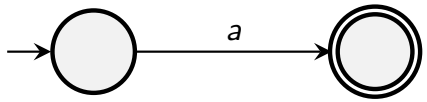
2. For any regular language L accepted by some DFA/NFA we can find a regular expression α that represents L (hard direction).



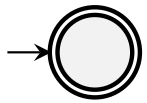
Regular expressions to regular languages (easy direction)

We use induction over the structure of regular expressions:

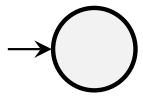
1. $L(a) = \{a\}$ is accepted by



2. $L(e) = \{e\}$ is accepted by



3. $L(\emptyset) = \emptyset$ is accepted by



4–6. For $L(\alpha \cup \beta)$, $L(\alpha\beta)$ and $L(\alpha^*)$ we use the closure properties of the regular languages.

The importance of finite automata

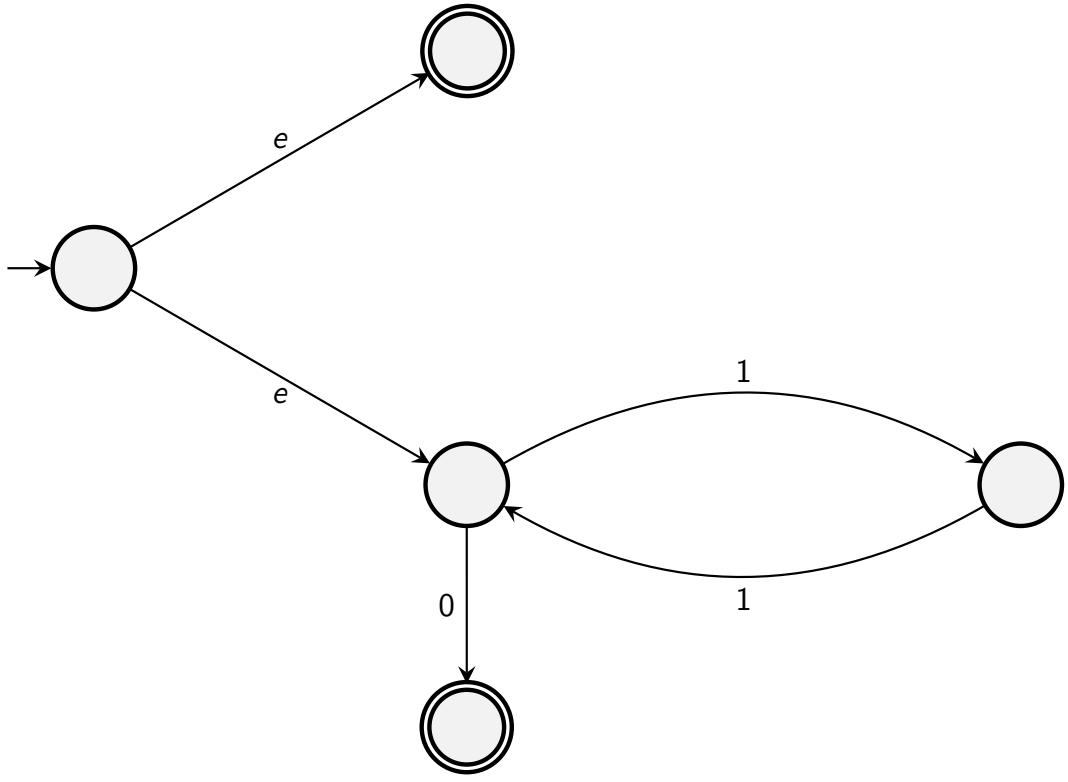
Finite automata can be used to check if a word fits the pattern described by a regular expression. They are used in all of the applications we listed before:

- Search and replace in a text editor.
- Validating user input in an application.
- Extracting information from a webpage e.g. topics, links, email addresses (can be done on a very large scale).
- Data processing and analysis (might want to find instances of a certain thing, eliminate noise).
- Searching for file types on an operating system.
- Syntax highlighting in an IDE.
- Lexical analysis as the first step of a compiler: Transforming a sequence of characters into a sequence of tokens.

Regex $\mapsto$ NFA in practice. First option: Just do it directly!

Regex: $e \cup (11)^*0$

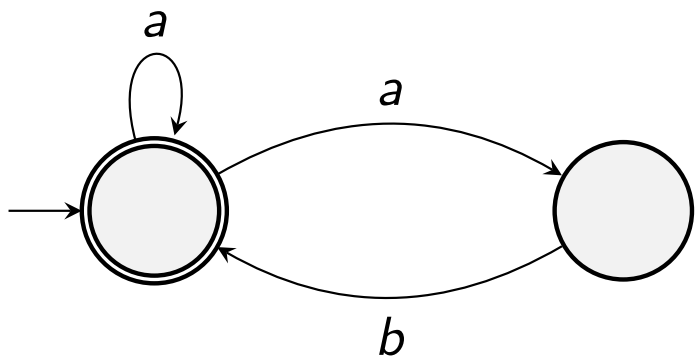
NFA:



Regex $\mapsto$ NFA in practice. Second option: Use formal constructions (but make sure you tidy up along the way!)

Regex: $(a \cup ab)^*$

NFA:



(but if we hadn't simplified each step the result would have been a mess!)

Something extra:

One of our former MSc students – Max Wheaton – invented Regex Tetris:

`https://steelwing158.itch.io/regetris`

You could do something similar in your Year 3 or MSc projects...